Building a Flexible Knowledge Graph to Capture Real-World Events

Laura Burdick
University of Michigan
wenlaura@umich.edu

Mingzhe Wang
Princeton University
mzwang@umich.edu

Oana Ignat
University of Michigan
oignat@umich.edu

Steve Wilson
University of Edinburgh
steven.wilson@ed.ac.uk

Yiming Zhang
University of Michigan
yimingz@umich.edu

Yumou Wei
University of Michigan
yumouwei@umich.edu

Rada Mihalcea
University of Michigan
mihalcea@umich.edu

Jia Deng
Princeton University
jiadeng@cs.princeton.edu

November 2019

1 Pipeline Overview

Events and situations unfold quickly in our modern world, generating streams of Internet articles, photos, and videos. The ability to automatically sort through this wealth of information would allow us to identify which pieces of information are most important and credible, and how trends unfold over time. In this paper, we present the first piece of a system to sort through large amounts of political data from the web. Our system takes in raw multimodal input (e.g., text, images, and videos), and generates a knowledge graph connecting entities, events, and relations in meaningful ways.

This work is part of the DARPA-funded Active Interpretation of Disparate Alternatives (AIDA) project[1], which aims to automatically build a knowledge base that can be queried to strategically generate hypotheses about different aspects of an event. We are participating in this project as a TA1 team, building the first step of the overall system.

Our approach is outlined in Figure 1 and will be discussed in detail in the following sections. The first step of the pipeline is pre-processing, shown in

[1] https://www.darpa.mil/program/active-interpretation-of-disparate-alternatives

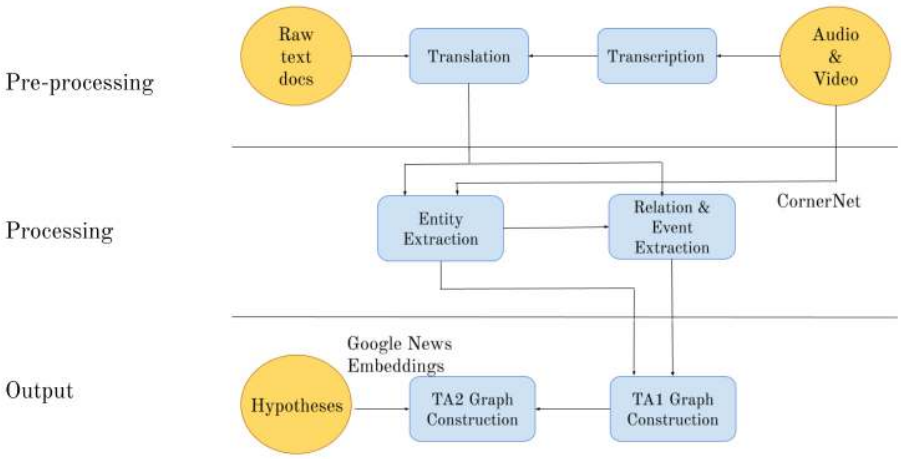


Figure 1: The overall AIDA pipeline. Our part of the system takes in raw text documents (top left) and audio and video (top right) and outputs a knowledge graph (bottom right).

the top row of Figure 1. Raw text documents, originally written in multiple languages, are translated into English, and audio and video clips are transcribed and translated into English. This translated data is passed to the second stage of the pipeline (middle row of Figure 1). Here, relevant entities (e.g., people, places, countries) are extracted, and these entities are used to extract relations and events linking the entities. Finally, these entities, events, and relations are passed to the last stage of the pipeline (bottom row of Figure 1). We output a fully-formed knowledge graph representing the information that we have gleaned from the raw input documents. This knowledge graph includes entities, as well as connections between them.

2 Data Pre-processing

We take in raw text, audio, and video as input. These files are pre-processed before we extract entities, events, and relations.

2.1 Translation

For each raw document, we detect the language of the text and translate non-English (Russian and Ukrainian) text into English. We convert each file from the LTF format to the RSD format, a format more suitable for translation as it removes unnecessary information.

We use an off-the-shelf translation tool to automatically detect the language

of the text data[2]. In comparison to other services, we found that the tool we use achieves qualitatively the best results.

After each text document is translated, we extract the entities, events and relations from them (discussed in Sections 3 and 4). Because this processing is done on translated documents, the preliminary output contains only English entities, events, and relations. However, we need entities, events, and relations in the original language of the document.

To back-align extracted elements into their original language, we translate each token back into its original language and search for its start and end offsets. For future work we plan to perform error analysis on our translation output. We are also currently exploring ways of working directly with the raw data, by using multilingual models (e.g., Multilingual BERT[3] [1]).

2.2 Transcription

Since the video files are mainly news-related, the knowledge and information captured in speech is essential to fully understand the video content. We extract a video transcript using speech-to-text processing. We extract audio from the video files and process the audio using an off-the-shelf transcription tool which supports transcription for English, Russian, and Ukrainian. These transcripts are then combined with normal text documents and processed in the same way.

3 Extracting Entities from Language

Once we have pre-processed the raw input data, we extract entities from text documents. Specifically, we use two entity extraction methods, and we combine the output from these methods to get a final list of entities.

3.1 Method 1: Ensemble Classifier

For our first method, we identify entities using string matching. We use two similarity metrics: cosine similarity and edit distance.

Cosine similarity, shown in Equation 1, is a measure of similarity between two vectors a and b. It can be interpreted as the angle between two vectors.

$$cos(a, b) = \frac{a \cdot b}{||a||\,||b||} \tag{1}$$

We first calculate word embeddings for all known entities in the training data and perform cosine similarity matching between all spans of tokenized text in the test data and all training entities. On the test data, we use a sliding window with a window size between one and five tokens to get all possible spans. While there are entities longer than five tokens in the annotation, they occur infrequently. To compensate for the fact that the sliding window misses

[2]For this submission we use the Google API, but any translation tool can be substituted.
[3]https://github.com/google-research/bert/blob/master/multilingual.md

Setting	Type Precision	Subtype Precision	Type Recall	Subtype Recall
High Recall	0.0790	0.0592	0.5817	0.4357
High Precision	0.1820	0.1382	0.4619	0.3508

Table 1: Precision and recall on AIDA entities using the ensemble classifier.

some long entities, we use spaCy [2] to parse all noun chunks and add them to the list of spans that are considered. Then, we extract as entities all text spans with a cosine similarity above a set threshold (we use 0.35) with a known training entity.

The second metric we use is edit distance, which measures the cost of transforming one string to another using the operations of insertion, deletion, and substitution[4]. If two strings have a low edit distance, they look similar to each other. Similar to the process of using cosine similarity, we measure the edit distance from all spans of tokenized words to known training entities, and extract as entities spans that have a low edit distance. We use a similarity ratio of 60% as the threshold for edit distance. This ratio can be understood as the edit distance normalized by the length of input strings.

We combine the above matching techniques using a weighted k-nearest-neighbor (KNN) classifier that takes a weighted vote of the top neighbors of the query [3, 4]. Typically, one span of text is matched to multiple known entities using both cosine similarity and edit distance. The KNN classifier leverages multiple potential matchings to maximize classification accuracy and make more robust predictions.

The KNN classifier takes an ensemble of the two matching techniques (cosine similarity and edit distance), as well as two off-the-shelf named entity recognition (NER) taggers, coreNLP [5] and spaCy [2]. By stacking these four classifiers and taking the union of their outputs, we are able to achieve better recall and precision than using any of the methods individually. Table 1 shows results using the KNN classifier. Here, we list precision and recall for both types and subtypes. Types are the main entity types (see Table 2), while subtypes are sub-categories within these main types.

There is a trade-off between precision and recall in our model, and by tuning parameters of the ensemble classifier, we are able to prioritize either precision or recall. Based on the nature of our problem, we are more interested in improving recall than precision, while not allowing precision to drop too low. Because of this, we use the "High Recall" setting. In this situation, it is better to have a larger knowledge base with some irrelevant items than to have a smaller knowledge base that is missing important entities and events.

3.2 Method 2: Contextualized Embedding Classifier

The second method that we consider for entity extraction is contextualized embeddings. Contextual embeddings give words different vector representations

[4]We use python module fuzzywuzzy to measure edit distance.

Entity Type	Definition and Examples
CARDINAL	Numerals that do not fall under another entity type
DATE	Absolute or relative dates or periods
EVENT	Named hurricanes, battles, wars, sports events
FAC	Buildings, airports, highways, bridges
GPE	Geo-political entities, countries, cities, states
LANGUAGE	Any named language
LAW	Named documents made into laws
LOC	Non-GPE locations, mountain ranges, bodies of water
MONEY	Monetary values, including unit
NORP	Nationalities or religious or political groups
ORDINAL	"first", "second"
ORG	Companies, agencies, institutions, etc.
PER	People, including fictional
PERCENT	Percentage (including "%")
PRODUCT	Vehicles, weapons, foods (not services)
QUANTITY	Measurements, as of weight or distance
TIME	Times smaller than a day
VEH	Vehicles
WEA	Weapons
WORK_OF_ART	Titles of books, songs

Table 2: Entity types for training contextualized embedding classifier.

depending on their contexts (e.g., the surrounding sentence). This allows the embeddings to capture high-level semantic features and often leads to better performance on downstream tasks. In this work, we consider the contextualized embedding algorithm BERT [1], though other algorithms could be easily substituted. BERT is particularly useful when there is not enough data to train a deep neural network from scratch. Our training data is too small to train a deep model without overfitting. Therefore, we leverage existing, pre-trained models that provide sufficient priors to successfully develop our own models.

We train our model on relevant entity extraction datasets, not including the AIDA training data. Specifically, we use OntoNotes [6] and REFLEX [7], datasets designed for NER. OntoNotes is an annotated collection of data from various sources such as news and telephone conversations. Its English corpus contains roughly 1.4 million words and covers a wide variety of entity types. The REFLEX dataset is smaller, containing about 22.5K English words, but it has data for an important entity type for our scenario, weapons. We merge the two datasets and work with the entity types shown in Table 2.

Our NER model architecture consists of BERT followed by a fully-connected layer as a classifier. We frame entity extraction as a multi-label, multi-class classification problem to account for the fact that an entity can have multiple labels (entity types). The standard "IOB" labeling scheme [8] is used to denote

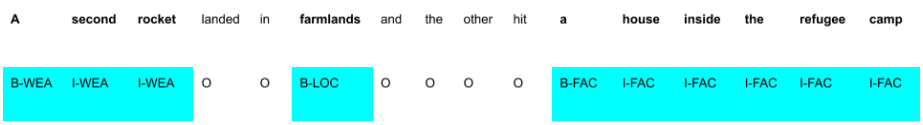


Figure 2: An example sentence with IOB-style annotations.

whether a token is inside an entity (I), outside of any entities (O) or at beginning of an entity (B). An example annotated sentence is shown in Figure 2. For each token, our model makes independent predictions about the probability that the token should be assigned each label.

3.3 Combining Entity Extraction Methods

Given a list of entities extracted using Method 1 and a list of entities extracted using Method 2, we combine these lists to get a final list of entities. First, we filter out entities that contain only punctuation, or entities that are stopwords (using the list of English stopwords from NLTK [9]).

Second, we combine entities from the two methods. We search through all of the entities, looking for overlapping text ranges between the two lists. Any entities that do not overlap with another entity are automatically added to the final list of entities. If there is overlap, then we look to see if the main type of the overlapping entities is the same. If the main type differs between the two methods, we discard these entities as inconsistent. If the main type is the same, we either keep the entity whose text range fully encompasses the other entity, or, if neither entity fully encompasses the other, we keep the entity from Method 2, contextualized BERT embeddings.

Finally, we do a second filtering pass to remove duplicate or near-duplicate entities. If a set of entities have overlapping text ranges, then we remove all but one entity from this set, according to the following rules: (1) we prefer entities that are longer sequences (at the word level), unless the entity is greater than five total words, and (2) we prefer entities that do not start or end with punctuation or stopwords[5], unless they are greater than three words longer than the next overlapping entity. This final list of entities is passed to the event and relation extraction section of the pipeline.

4 Extracting Events and Relations from Language

Given a set of entities of interest, we then use the parsed language output to infer events and relations as outlined in the seedling ontology. Two approaches are considered to obtain coarse labels for the possible relationships between the

[5]Here, we use a smaller set of stopwords: the a an for and nor but or yet so at to.

entities: a graph-based and a span-based approach. Figure 3 illustrates the steps of this process.

4.1 Graph-Based Approach

First, we construct a small graph, $G_S = (V_S, E_S)$ for each parsed sentence S, where $V_S = \{w|w \in S\}$ and w is a word. The graph's edges, E_S, are defined by the set of dependency links between words (vertices) as well as adjacency links that occur between two words, w_1 and w_2 (an example dependency parse is shown in Figure 3a). If w_2 immediately follows w_1 in S and both w_1 and w_2 belong to the same noun phrase or verb phrase as identified by the syntactic constituency parse of S, then we add an adjacency link between them (example constituency parses and connected graphs are shown in Figures 3b and 3c). Then, to identify relationships between entities, we consider the set of vertices corresponding to words that are part of a given entity phrase and explore all incoming and outgoing edges. If any edge links to a vertex that represents a words belonging to a different entity of interest, we determine that there is a relationship between these two entities. If the connecting edge is a dependency edge, we use the dependency label as a coarse initial label for the relationship. Otherwise, if the edge is phrase-based (i.e., the two entities are part of a larger identified phrase), we use the span of words between the entities as a *plain text label* for the relationship.

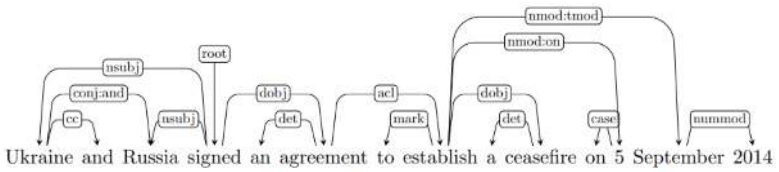
4.2 Span-Based Approach

In this alternative method, we rely on the ordering of words in the parsed sentences to naturally convey information about the relationships between entities. Given two entities that appear in the same sentence, we extract the span of words between the end of the first entity and the start of the second entity. We do not consider spans that are too long (we define this as a 15+ word span) or those that contain end-of-sentence markers, since these likely do not capture a meaningful connection between the two entities. All other spans are treated as a relationship between the two entities, with the span itself being treated as a *plain text label*.

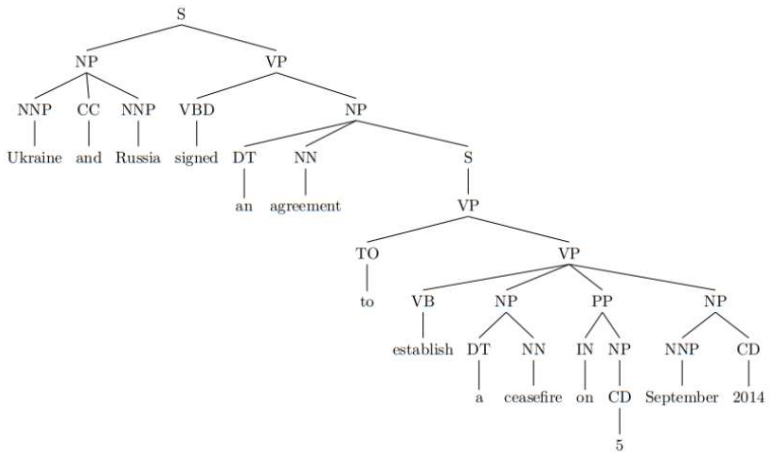
After identifying pairs of entities and *plain text labels*, we use a nearest-neighbor matching approach to predict the relationship or event label from the ontology, as described in the next section.

4.3 Labeling Relations and Events

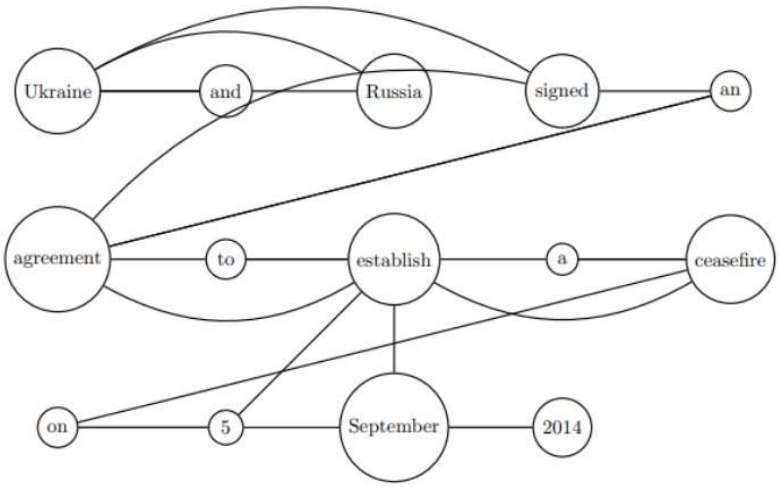
To assign relation and event types from the ontology to the extracted keywords, we first filter the possible event and relation types based on the possible valid argument types for the event or relation. We take the types predicted for each entity involved in the event or relation and then remove any event or relation type which does no allow for these entity types.



(a)



(b)



(c)

Figure 3: Steps in the event / relation extraction process for the sample sentence "Ukraine and Russia signed an agreement to establish a ceasefire on 5 September 2014". (a) The dependency graph for this sentence. (b) The constituency graph for this sentence. (c) Connected graph for this sentence. The first two graphs were generated using the Stanford Parser [10].

Then, as was done with entity types, we use a nearest-neighbor approach based on word embedding similarity, using the GloVE word embeddings [11]. Since each relation or event is a short phrase of text, we compute its embedding as the average of word embeddings:

$$E(x) = \frac{1}{N} \sum_{w} E(w) \tag{2}$$

where $E(\cdot)$ is the embedding function, w is a word in the relation or event x and N is the number of words in the span of text marked as the relation or event. Each event or relation type has an associated description in the ontology. Using this description, we create an embedding of the each event or relation type T using Equation 2. Then, each relation or event is assigned to a type, whose embedding is the closest to that of the relation or event, determined by cosine similarity:

$$Category(x) = argmax_T(cos(E(C), E(keyword))) \tag{3}$$

where only types that were not previously filtered out are considered.

5 Extracting Entities from Images and Videos

The development of deep learning has resulted in successful object detectors to identify visual objects in images and videos. We apply the CornerNet [12] detector to generate bounding boxes as candidates of semantic entities. For videos, bounding boxes are generated from key frames. Our CornerNet detector is fine-tuned on the AIDA domain. Since we found few images for some expected categories, such as machine gun and protester, we fine-tune our detector with a hierarchical relational loss to alleviate the imbalance of training data. Fine-tuning is described in Section 5.1.

After generating bounding boxes with the object detector, to filter less important boxes and map them onto the fine-grained entries in the ontology, we incorporate the textual background that is relevant to this visual document element. For an image, such background refers to the textual document elements which share the same document ID as this image. For a video, we consider transcripts generated from speech-to-text as its background.

Textual entities are extracted from the background using the approach described in Section 3. Bounding boxes are then mapped onto these textual entities. For a box b with category label l and a textual entity e whose name is x and type is y, the matching score $s(b, e)$ between b and e is calculated as

$$s(b, e) = \frac{1}{2} \|w_l\| (\|w_e + e_y\|) \tag{4}$$

where w_x is the GloVe word embedding of x [11].

Each box b is matched with the entity e to maximize the matching score $s(b, \cdot)$. The name and type of e is also passed to b. They are merged as the

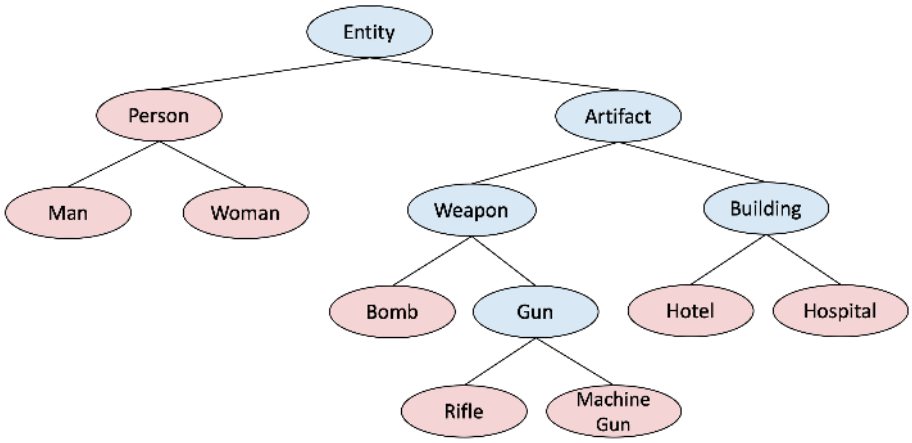


Figure 4: A subset of the category syntex tree. Red nodes are target categories for finetuning. Buld nodes are other internal nodes added from WordNet.

same knowledge entity in our AIF graph and bounding box b is added as an image/keyframe justification. We filter the boxes whose highest matching scores are lower than a threshold.

5.1 Fine-tuning CornerNet Object Detector

To adapt CornerNet for AIDA, we collect the 47 most frequent entity types from the AIDA training data and fine-tune CornerNet for these 47 categories.

Training images for these categories are extracted from the Visual Genome [13] dataset with bounding boxes annotations. Since some categories, such as missile and machine gun, appear rarely in daily life images, we found very few annotations for these less-common categories. To deal with this imbalance in the training data, we apply similar training methods as YOLO9000 [14] to learn from category relations.

We build a syntax tree of the 47 training categories following the WordNet hyponym hierarchy [15]. A subset of our category tree is displayed in Figure 4.

Given this category tree, each bounding box is re-annotated as all categories that are the parents of its original category. For example, "hotel" box is also labeled as "building", "artifact", and "entity".

The original classification problem, to classify a box into 47 categories, is decomposed into several small classification tasks, one for each internal node along the path from its label up to the root. For each internal node, the model classifies this box among its children. For example, to classify a box as a bomb, the model needs to classify it between {person, artifact}, {weapon, building} and {bomb, gun}.

Our detector is trained with the cross-entropy loss of each small classification. For an image that has the label "bomb", it contributes to three small

classification losses as mentioned above.

6 Graph Construction

Our goal for the output of our system is to create knowledge graphs with relevant and useful entities, connected by relations and events. We store this graph in Argument Interchange Format (AIF), a common format for storing arguments and supporting evidence [16]. We create a single knowledge graph for each input document, and we do not link entities across documents.

For each document, we create an initial graph by encoding the set of entities, relations, and events extracted for that document. Entities that are associated with identical ontology categories and strings (excluding stopwords and differences in capitalization) are clustered together to form an entity cluster. Entity names are truncated at 100 characters to ensure that they fit in the AIF format. For events and relations, confidence is calculated by taking the average of the confidence of the entities associated with the event or relation.

Justifications are included in the AIF graph for all entities, events, and relations. For elements found in the text documents, justifications are textual spans. For elements found in the audio and video documents, justifications are bounding boxes. Multiple justifications are included, where appropriate, for a single element, but one justification is marked as informative.

Optionally, we include JSON-serialized embeddings for each entity extracted from video in the knowledge graph. For each bounding box in a video frame, CornerNet has embeddings for the two corners defining these box. We combine these two corner vectors as the embedding for this box and the entity labeling the box. This information can be used further along in the overall system to compare entities and learn relationships between entities.

6.1 Hypothesis-Adjusted Knowledge Graph

We are also able to adjust our knowledge graphs given specific hypotheses about the events covered by the knowledge graph. A hypothesis comes in the form of an small AIF graph with entities and their hypothesized relationships. Assuming that this hypothesis is accurate, we use this information to update the justifications in our knowledge graph. This allows us to flexibly update our graph to reflect the current belief of the situation.

Specifically, to update our knowledge graph, we extract all of the entity names and text strings from the hypothesis graph, and we map each of these names to embeddings. (We use Google News embeddings, but any embeddings could be substituted.) We combine these hypothesis embeddings to create a final embedding h_{emb} by taking the maximum value from each embedding dimension, as shown in Equation 5. Here, H is the set of text values contained in the hypothesis graph, $emb(h)$ is the embedding of the text value h, and max is the element-wise maximum operation.

$$h_{emb} = max_{h \in H}(emb(h)) \tag{5}$$

To adjust the confidence scores in our graph, for each entity E in our graph, we look at all text values e associated with E. Similarly to our calculation of h_{emb}, we take the element-wise maximum of the embedding dimensions of the embeddings for $e \in E$, as shown in Equation6.

$$e_{emb} = max_{e \in E}(emb(e)) \tag{6}$$

Finally, we take the cosine similarity of h_{emb} and e_{emb} and use that value to adjust the entity confidence c, as shown in Equation 7.

$$c = 1.0 - \frac{t_{emb} \cdot h_{emb}}{\|t_{emb}\| \|h_{emb}\|} \tag{7}$$

The adjusted graph has the same set of entities, relations, and events, but the confidences are adjusted to take into account the hypothesis graph that we are given.

7 Conclusion

In this project, we build a complete pipeline to extract entities, relations and events from multimedia input including text, videos and audio. The key steps of this pipeline are entity extraction from both languages and images, and event and relation extraction. Text entity extraction is addressed using an ensemble classifier and contextualized embeddings, while visual entity extraction uses CornerNet to detect objects. We additionally extract relations and events from text using graph-based and span-based approaches. Finally, a cohesive knowledge graph including entities, events, and relations is output.

One of the challenges we encountered is balancing precision and recall in entity extraction. While there are many entities present in the test, not all of them are relevant to the events that we are seeking to capture in our knowledge graph. We address this challenge by focusing our entity extraction on certain categories present in the AIDA ontology, as well as tuning our model parameters to achieve a balance between precision and recall.

8 Acknowledgements

This material is based in part upon work supported by the DARPA AIDA program under grant #FA8750-18-2-0019. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the Defense Advanced Research Projects Agency.

References

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *NAACL-HLT*, 2019.

[2] M. Honnibal and I. Montani, "spaCy 2: Natural language understanding with Bloom embeddings," *Convolutional Neural Networks and Incremental Parsing*, 2017.

[3] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inf. Theor.*, vol. 13, no. 1, pp. 21–27, Sep. 2006. [Online]. Available: https://doi.org/10.1109/TIT.1967.1053964

[4] R. J. Samworth *et al.*, "Optimal weighted nearest neighbour classifiers," *The Annals of Statistics*, vol. 40, no. 5, pp. 2733–2763, 2012.

[5] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky, "The Stanford CoreNLP natural language processing toolkit," in *Association for Computational Linguistics (ACL) System Demonstrations*, 2014, pp. 55–60. [Online]. Available: http://www.aclweb.org/anthology/P/P14/P14-5010

[6] R. M. Weischedel, E. H. Hovy, M. P. Marcus, and M. Palmer, "Ontonotes : A large training corpus for enhanced processing," 2017.

[7] C. Walker, S. Chen, S. Strassel, J. Medero, and K. Maeda, "Reflex entity translation training/devtest ldc2009t11," 2009.

[8] L. Ramshaw and M. Marcus, "Text chunking using transformation-based learning," in *Third Workshop on Very Large Corpora*, 1995. [Online]. Available: https://www.aclweb.org/anthology/W95-0107

[9] E. Loper and S. Bird, "Nltk: the natural language toolkit," *arXiv preprint cs/0205028*, 2002.

[10] D. Chen and C. Manning, "A fast and accurate dependency parser using neural networks," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 740–750.

[11] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[12] H. Law and J. Deng, "Cornernet: Detecting objects as paired keypoints," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 734–750.

[13] R. Krishna, Y. Zhu, O. Groth, J. Johnson, K. Hata, J. Kravitz, S. Chen, Y. Kalantidis, L.-J. Li, D. A. Shamma, M. Bernstein, and L. Fei-Fei, "Visual genome: Connecting language and vision using crowdsourced dense image annotations," 2016. [Online]. Available: https://arxiv.org/abs/1602.07332

[14] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7263–7271.

[15] C. Fellbaum, "Wordnet," in *Theory and applications of ontology: computer applications*. Springer, 2010, pp. 231–243.

[16] I. Rahwan and C. Reed, "The argument interchange format," in *Argumentation in artificial intelligence*. Springer, 2009, pp. 383–402.